Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL FINAL DE GRAU

TÍTOL DEL TFG: Automation Software Develoment

TITULACIÓ: Grau en Enginyeria de Sistemes de Telecomunicació

AUTOR: Ivan Tosso Tacón

DIRECTOR: Albert Tort Pugibet

SUPERVISOR: Dolors Royo Valles

DATA: 20 de agost del 2017

Resumen

Actualmente muchas empresas del sector tecnológico, se dedican al desarrollo de software para cubrir necesidades internas o de terceros El desarrollo de software es cada vez más extendido en más ámbitos de la sociedad, se ejecuta en una diversidad mayor de dispositivos y se aplica en cada vez más dominios.

El software desarrollado requiere la aplicación de actividades de Software Quality Assurance (QA), con el objetivo de comprobar **criterios de calidad**. Una de las actividades principales de QA es el *testing* de software. El *testing* se basa en la aplicación de casos de prueba que deben diseñarse con criterio a partir de los requisitos. Existen varias formas de especificar requisitos y en las metodologías ágiles, un artefacto esencial son las historias de usuario, que se pueden especificar usando lenguajes como **Gherkin** y automatizados con herramientas como **Cucumber**. Por otro lado, existen distintos niveles de prueba, des de casos unitarios a pruebas de aceptación funcional End-to-End.

El propósito de este proyecto, consiste en agregar a una herramienta comercial de Sogeti, denominada **kCycle**, que genera pruebas end-to-end a partir de historias de usuario y modelos de flujo, una funcionalidad que facilite la generación automática de estas pruebas en formato **Gherkin**, en un proyecto **Maven** compatible con el entorno de desarrollo Eclipse y que permita la utilización de **Cucumber** para asistir la automatización de las pruebas.

Abstract

Currently many companies in the technology sector are dedicated to the development of software to meet internal or third-party needs. Software development is increasingly widespread in more spheres of society, runs on a greater diversity of devices and is applied every time More domains.

The software developed requires the application of Software Quality Assurance (QA) activities, in order to check quality criteria. One of QA's core activities is software testing. The testing is based on the application of test cases that must be designed with criteria based on the requirements. There are several ways to specify requirements and in agile methodologies. An essential artifact are user stories, which can be specified using languages such as Gherkin and automated with tools such as Cucumber. In testing there are different levels of test, from unit cases to End-to-End functional acceptance tests.

The purpose of this project is add in a Sogeti business tool, called kCycle, which generates end-to-end tests from user stories and flow models, a functionality that facilitates the automated generation of these tests in format Gherkin, in a Maven project compatible with the Eclipse development environment and allowing use Cucumber to assist the automation of testing.

Agradecimientos

Quiero agradecer a Dolors su ayuda en la comunicación con la universidad y el soporte recibido en todo momento.

También quiero agradecer a mi director Albert Tort por la oportunidad de trabajar en este proyecto, por las cosas que he podido aprender y la experiencia en la planificación y ejecución de proyectos que he podido obtener.

Además, quiero agradecer a mis compañeros y amigos Cristian y Ángel por el soporte recibido en los inicios sobre todo cuando me sentía perdido en el desarrollo y en los últimos días por las ayudas en cuanto al correcto funcionamiento de la herramienta sobre la que he desarrollado nuevas funcionalidades.

Por último y no menos importante, quiero agradecer a Lydia por su apoyo moral y empuje para finalizar el trabajo además del apoyo en todo este camino.

Tabla de contenido

Figuras

Tablas

1. Introducción

Este Trabajo de Final de Grado “Automated Project Development” ha sido realizado en cooperación con la empresa *Sogeti[1]*, una empresa dedicada a ofrecer servicios de *testing* a clientes del sector de las TIC.

Concretamente, el proyecto tiene como objetivo la adición de una funcionalidad a una herramienta comercial de Sogeti para permitir la creación automática de proyectos de automatización de *testing* de software que contemplen los casos de prueba generados por dicha herramienta. Por tanto, este proyecto contribuye directamente a kCycle, una herramienta desarrollada en el marco del departamento de I+D de la empresa.

1.1 Formulación del problema

En el estado actual de desarrollo de software, las empresas desarrolladoras antes de sacar el producto a un mercado tan saturado donde la única diferenciación entre aplicaciones es la calidad de estas.

Por tanto, una vez los desarrolladores han avanzado lo suficiente para empezar a probar la calidad de sus productos, contratan a empresas que les hagan estos test de QA teniendo en cuenta diferentes métricas. Por lo general, estas comprobaciones son de carácter manual, esto implica que un equipo de *testing* de empiezan a hacer **pruebas manuales**[2] con todas las plataformas que requiera el cliente desde Web hasta alguna App para android o IOS.

Con el paso del tiempo, esta tendencia está cambiando ya que ahora se busca a un equipo que utilizando Selenium, automatice estas pruebas ya que el servicio debe estar activo en todo momento y cualquier fallo puede ser determinante. El problema empieza cuando no todas las empresas tienen este perfil de QA y se hace difícil el poder encontrar equipos que realicen el trabajo ofreciendo una integración y un mantenimiento de este.

1.2 Objetivos del trabajo

Esta idea nace con la finalidad de conseguir una mayor eficiencia de recursos al realizar el *testing* de una aplicación y poder centralizar los datos resultantes en el control de la ejecución de pruebas y el análisis de los veredictos, nace este proyecto, basado en la utilización de un lenguaje entendible por todos los roles (desarrolladores, testers, responsables de negocio,…) implicados en la calidad del software. En este contexto, el proyecto sigue la aproximación BDD, (Behaviour-Driven Development).

Para poder entender la motivación de este proyecto, hay que entender que el proceso de desarrollo BDD pretende realizar un acercamiento entre la parte técnica y la de negocio a partir de un desarrollo agile, a través de la

utilización de una metodología Scrum y haciendo uso de un lenguaje de programación apto para todos los roles de este escenario.

Una vez definidos los pasos a seguir, lo primero que se nos viene a la cabeza para acercar a todos los actores de este proceso, es utilizar un lenguaje natural, un lenguaje que puedan entender desarrolladores, roles de negocio, responsables de calidad, etc.

En este punto es donde interviene Gherkin, un lenguaje creado para llevar a cabo este acercamiento entre todas las partes afectadas. A través del uso de **Keywords**[3], podemos hacer sentencias para poder llevar a cabo el desarrollo de los **criterios de aceptación**[4] del software en cuestión.

Por tanto, el objetivo final de este proyecto es la creación de una funcionalidad que permita exportar los **casos de prueba end-to-end[5]** introducidos en la herramienta **kCycle** a un proyecto de **Cucumber** de forma automática, capaz de ser importado en cualquier IDE creado a partir de Eclipse.

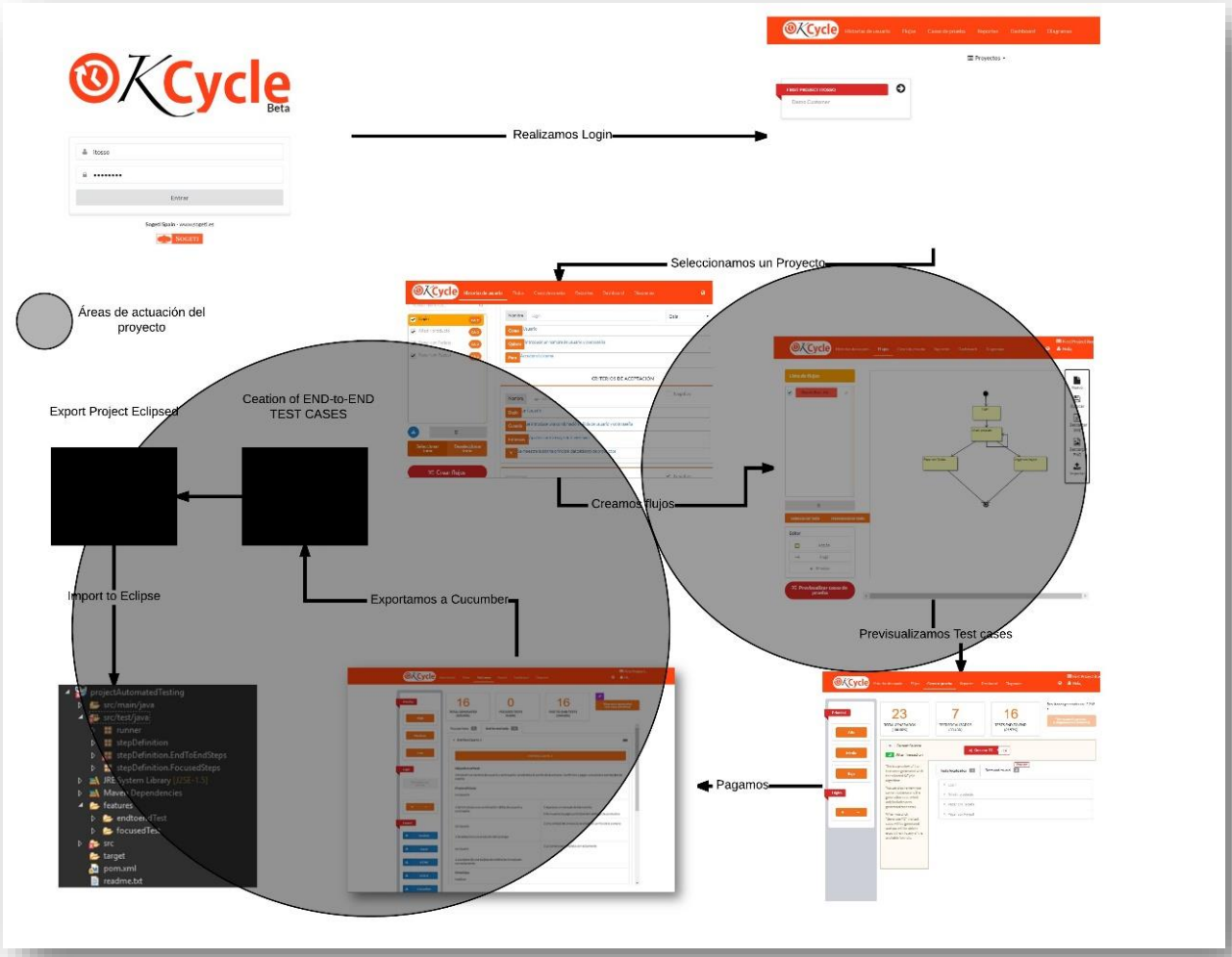

Figura 1 Diagrama de flujo de kCycle

1.3 Introducción a kCycle

KCycle es una aplicación propietaria de la empresa Sogeti que surge como solución a la necesidad de la creación de casos de prueba por parte de los clientes utilizando el **lenguaje natural**.

Para lograr el objetivo, kCycle se basa en la aproximación **Behaviour-Driven Development (BDD)[6]** habitualmente aplicado en entornos de desarrollo ágil. Esta aproximación tiene como base la especificación de las **historias de usuario [7]**, que son una representación ágil de los requisitos del software, junto con los criterios de aceptación de estas historias de usuario (HU) y de los flujos del software. kCycle es capaz de crear casos de prueba end-to-end y exportarlos en diferentes formatos. Esto es posible a partir de la definición de HU (focalizadas en funcionalidades concretas) y de modelos de los flujos posibles que combinen dichas historias de usuario.

2. Contextualización

2.1 Estado del ARTE

En este apartado, se muestra la literatura sobre los temas relacionados con el objetivo del proyecto. Dado que el estudio está basado en la creación de proyectos en *Cucumber* y que este *framework* se basa en la creación de sentencias en **Gherkin** y los métodos utilizados por Java y las librerías de **Selenium**, este apartado puede abarcar diversos ejes:

- Trabajo previo sobre Gherkin y sus funcionalidades.
- Trabajo previo sobre Java y librerías de Selenium.
- Trabajo previo sobre funcionamiento de Cucumber.
- Estudio sobre model-driven testing y su uso profesional.

2.1.1 *Trabajo previo sobre Gherkin y sus funcionalidades*

Cuando hablamos de **Gherkin**, hablamos de un lenguaje creado para poder crear sentencias en un lenguaje comprensible para personas no técnicas ni entendidas en el desarrollo software. Si nos centramos en su funcionamiento, **Gherkin** utiliza *Keywords* para crear una relación entre un lenguaje natural y un lenguaje comprensible por compiladores. A través de librerías de Cucumber y/o librerías de propias de Gherkin, se ofrece este soporte para poder hacer más comprensible el desarrollo de aplicaciones.

2.1.2 *Trabajo previo sobre java y librerías de Selenium*

Cuando nos referimos a JAVA, hablamos de un lenguaje muy amplio y extendido en el mundo de la programación, por tanto los recursos y las librerías que podemos encontrar abarcan prácticamente todo el marco de la desarrollo software actual. Por tanto, **Selenium[8]** es una librería más a la que tenemos acceso cuando pretendemos realizar acciones que requieren el control de navegadores web así como Chrome o Firefox entre otros.

Gracias a esta librería, podemos obtener el control completo de un navegador y realizar las acciones que creamos conveniente. Estas acciones comprenden un abanico muy amplio de soluciones a cualquier tarea que debamos hacer desde maximizar la pantalla a obtener el control cualquier elemento que se encuentre en la pantalla.

2.1.3 *Trabajo previo sobre funcionamiento de Cucumber*

La idea de poder procesar un lenguaje natural nace con la intención de adecuar un vocabulario para poder obtener una comunicación más efectiva. De esta gran idea surge Cucumber, un **framework[9]** que realiza la unión de un lenguaje natural y un lenguaje comprensible por las computadoras.

Por tanto, *Cucumber* hace de intermediario entre humanos y computadores. La forma que tiene de hacernos de intermediario es mediante el uso de lenguajes muy extendidos como Java, JavaScript o Ruby entre otros.

A partir de las HU creadas por los PO (*Product Owner*), *Cucumber* nos crea los métodos en el lenguaje extendido que se utilice para que los compiladores sean capaces de entender las sentencias y poder realizar las sentencias que permiten ejecutar automáticamente los casos de prueba que se programarán.

Cabe destacar que Cucumber en ningún caso está pensado para casos de prueba del tipo End to End. Su principal función es ayudar en pruebas unitarias y no en el test de un proceso, que no es más que la unión de diferentes casos de una forma lógica con la finalidad de testear diferentes acciones unidas con un sentido y una finalidad concreta.

2.1.4 *Estudio sobre model-driven testing y su uso profesional*

El Model-Driven Testing (MDT) incluye las distintas técnicas y procedimientos automáticos o semiautomáticos que tienen como objetivo, a partir de un modelo del sistema, generar casos de prueba que prueben dicho sistema. A grandes rasgos cuando se habla del modelo del sistema, nos referimos a una especificación formal de los requisitos funcionales del sistema. La generación automática de pruebas se basa en criterios de cobertura previamente establecidos o parametrizados.

Actualmente, aunque existen técnicas diversas [...] de Model-Driven Testing a partir de modelos de requisitos (casos de uso, diagramas de actividad...) no existen herramientas comerciales similares a KCycle que permitan generar casos de prueba a partir de historias de usuario y flow sketches. Tampoco existen herramientas que permitan generar la estructura de un proyecto de automatización de los diseños de prueba generados. En este punto, kCycle, junto con el proyecto presentado, ofrece esta ventaja singular.

2.1.5 *Conclusión*

Después de hacer un análisis del estado del arte, para conseguir esta nueva funcionalidad deberemos entender cuáles son los pasos que sigue Cucumber para hacer esta conexión entre ambas partes y de que forma la realiza.

El problema principal es la creación del mecanismo que extrae los casos de prueba End to End ya que Cucumber no está preparado para este tipo de pruebas. En este aspecto, se da un valor añadido a kCycle ya que se ofrece una solución a un problema que afecta a los usuarios de Cucumber, que no tiene soporte para realizar este tipo de pruebas en un entorno de QA

2.2 Alcance del proyecto

Este proyecto abarca las siguientes funcionalidades concretas:

- Implementación de un mecanismo para obtener los datos de creados por kCycle y escribirlos en archivos con una extensión entendible por Cucumber.
- Creación de proyectos de Cucumber con la estructura utilizada por **Maven[10]** y los archivos necesarios para poder ser funcional.
- Integración de las dos funcionalidades anteriores en la herramienta kCycle.

2.2.1 *Objetivos*

El objetivo final de este proyecto, es la creación de un botón con una lógica que permita a los usuarios de **kCycle** exportar un proyecto de **Maven** utilizando la estructura que ofrece *Cucumber* poder realizar la automatización de pruebas utilizando un lenguaje apto para personas no técnicas además de ofrecer una solución para los problemas de cucumber sobre los test del tipo End-to-End.

2.2.2 *Posibles obstáculos*

Hay varios puntos que pueden afectar al desarrollo del proyecto, ya que la estructura del proyecto tiene una lógica que no hay que obviar.

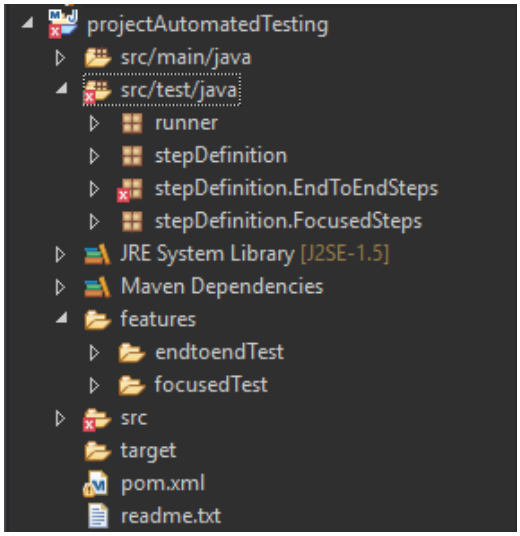

Figura 2 Distribución de carpetas en el proyecto de Cucumber.

Dentro de un proyecto de Cucumber, hay varios tipos de archivos esenciales para el correcto funcionamiento del proyecto:

- **FEATURE** files: Estos son los archivos donde se escriben las Historias de Usuario para describir los pasos a realizar y los criterios de aceptación. Los criterios de aceptación son en sí los *Steps* que hay que realizar para comprobar los requisitos especificados.

- **Step Definition:** Son archivos **JAVA** donde están definidos los pasos especificados en **Gherkin.** En estos archivos es donde los criterios de aceptación están especificados en JAVA. Para hacer la traducción de Gherkin a JAVA, internamente utiliza librerías que hacen esta conversión. Hay que vigilar ya que tendremos que utilizar la misma metodología para el correcto funcionamiento en la ejecución.

- **POM.xml**: Aquí es donde se definen las dependencias de las librerías que va a utilizar MAVEN. Cada vez que se define una nueva dependencia MAVEN busca en el repositorio que hemos definido y descarga las librerías para poder hacer uso de ellas.

- **Runner**: Clase JAVA en la que definimos donde se encuentran los archivos FEATURE y las *'Step Definition'.* Si la ruta donde se encuentran estos archivos no es la correcta, no podremos realizar ninguna ejecución.

Además de asegurarnos de que estos archivos son creados correctamente, hay que tener en cuenta que los archivos FEATURE exportados, son de las HU seleccionadas por el usuario en el proyecto desde el que se trabaja.

2.3 Metodología y Rigor

2.3.1 *Métodos de trabajo*

El desarrollo del proyecto se ha llevado a cabo mediante **SCRUM[11]** una metodología ágil, debido a que es la estrategia seguida por el equipo de *kCycle*. Esta forma de trabajar permite un mayor control a través del feedback del responsable del proyecto.

Se incluirán tres iteraciones de una duración determinada (4 semanas), obteniendo un producto fiable y funcional al finalizar cada iteración y por tanto, entregables con nuevas características.

La **primera iteración**, se estudió el funcionamiento de *kCycle* y en el mecanismo para la obtención de las historias de usuario y los criterios de aceptación impuestos por el usuario. Esta iteración era primordial ya que los demás módulos dependen de conseguir esta funcionalidad.

En la **segunda iteración**, se centró en estudia la mejor manera de implementar la lógica en el proyecto de *Cucumber*. Se decidirá cuál es la mejor distribución de carpetas dentro del proyecto, para hacerlo más intuitivo de cara al usuario final. Finalmente se llevará a cabo el desarrollo del módulo.

En la **tercera iteración**, se completó el proyecto incorporando esta nueva funcionalidad a la versión de producción, corrigiendo los posibles errores que hayan surgido y llevando a cabo pruebas de validación del proyecto.

2.3.2 *Herramientas de seguimiento*

Debido a que el trabajo a desarrollar forma parte de un proyecto comercial que está en constante desarrollo por parte de un equipo, se utilizará un repositorio alojado en GitHub para el control de versiones y Git como herramienta de gestión del repositorio; esto va a garantizarnos una disponibilidad del código en cualquier plataforma y facilidad de recuperar el código anterior en caso de errores.

El seguimiento del trabajo se refleja a través de Trello, un sistema de tickets donde se reúnen las tareas a realizar y a medida que finalizamos las tareas se irán marcando como ‘Completadas’, lo que aporta una mayor precisión del estado del proyecto.

2.3.3 *Métodos de validación*

Para validar los avances que se van produciendo, van a utilizarse los siguientes métodos:

- Se establecerán reuniones con el responsable del proyecto para saber el estado de los avances, se hará mediante reuniones cada dos semanas en las que se mostrarán los avances logrados hasta el momento de la reunión. Estas reuniones nos servirán para recibir un *feedback de los avances logrados.*

- Teniendo como objetivo la comprobación de que las funcionalidades desarrolladas tienen un resultado positivo, se incluirá una fase de testing, la cual nos permitirá saber si una funcionalidad está operativa antes de pasar a la siguiente.

- Para asegurarnos que la validación temporal se ajusta a la realidad, se tomará nota del estado de las tareas y de su duración. Esto nos permitirá saber que tareas son más susceptibles a cambios y los posibles motivos que nos llevan a estos cambios.

2.3.4 *Entorno de desarrollo*

En *kCycle*, la forma de trabajo es colaborativa y por ello es preciso el uso de herramientas que permitan tanto el trabajo individual para crear una nueva funcionalidad como el trabajo compartido, para agrupar un conjunto de nuevas funcionalidades de interactuación.

En este proyecto, las herramientas principales utilizadas son las siguientes:

- **Eclipse:** Es una *IDE (Integrated Development Environment)* que permite desarrollar código en cualquier tecnología, principalmente Java o relacionados.

- **Spring:** Framework que posibilita ejecutar instancias de aplicaciones Java. Permite la integración con Eclipse y la ejecución en un servidor alojado en el ordenador personal.

- **Tomcat:** Implementación de código abierto de Java Servlet, JavaServer Pages, Java Expression Language y Java WebSocket. Utilizado como servidor en aplicaciones Web.

- **Git:** Programa que permite la gestión de repositorios de código. En *kCycle,* se utiliza para mezclar las diferentes funcionalidades implementadas por diferentes desarrolladores.

- **Jenkins:** Aplicación web que permite compilar versiones, generar *builds* y ejecutar pruebas automáticas sobre esta.

3. Análisis de requisitos

En este apartado, hay una especificación de los requisitos que debe cumplir el proyecto, cuyo objetivo es obtener un proyecto para ser importado a cualquier IDE de eclipse.

3.1 Requisitos funcionales

Los requisitos funcionales, son los que describen las funcionalidades que ofrecerá el sistema y cómo se comportará ante un conjunto de entradas y cada uno estará descrito por los siguientes campos:

- **Descripción:** describe en que consiste el requisito.
- **Justificación:** indica el motivo por el cual se añade este requisito.
- **Criterio de satisfacción:** indica cómo se puede comprobar que el requisito se ha alcanzado con satisfacción.
- **Prioridad:** muestra el grado de prioridad del requisito valorado como Bajo, medio o Alto.
- **Satisfacción del cliente:** indica el grado de satisfacción que provoca sobre el cliente el cumplimiento del requisito, valorado con una escala ascendente entre 1 y 5.
- **Insatisfacción del cliente:** indica el grado de insatisfacción que provoca sobre el cliente el cumplimiento del requisito, valorado con una escala ascendente entre 1 y 5.

Tabla 3.1 Requisito del botón de exportación

Requisito #1	**Botón de exportación a Cucumber**				
Descripción	Se trata de un botón en la pestaña de test cases. Este botón permite la descarga de un proyecto de Maven utilizando Cucumber para tratar la automatización de pruebas desde un punto de vista basado en BDD.				
Justificación	Sin la completa funcionalidad del botón, no será posible la descarga del proyecto y por tanto la impresión del usuario hacia kCycle será muy negativa y puede incidir directamente en futuros clientes de esta herramienta.				
Prioridad	Alta	**Satisfacción del cliente**	5	**Insatisfacción del cliente**	5

Tabla 3.2 Requisito de adquisición de datos

Requisito #2	Adquisición correcta de HU y criterios de aceptación				
Descripción	Se trata de que el usuario adquiera los archivos FEATURE correspondientes a las HU que había seleccionado previamente.				
Justificación	Es la principal funcionalidad, el poder exportar las Historias de Usuario a un lenguaje de programación que puedan resolver los IDE creados a partir de Eclipse en un lenguaje natural.				
Prioridad	Alta	**Satisfacción del cliente**	5	**Insatisfacción del cliente**	5

Tabla 3.3 Requisito de generación correcta de clases JAVA

Requisito #3	Generación correcta de las clases a partir de las HU				
Descripción	Se trata de que el proyecto generado, utiliza una conversión interna para cambiar de lenguaje natural de Gherkin a una clase JAVA que sea entendida por los compiladores.				
Justificación	Es posible que de todo el proyecto, los usuarios solo requieran los archivos FEATURE para poder añadirlos a su proyecto, sin la necesidad de utilizar las clases de JAVA creadas.				
Prioridad	Alta	**Satisfacción del cliente**	5	**Insatisfacción del cliente**	5

3.2 Requisitos no funcionales

Los requisitos no funcionales son aquellos que no cubren ninguna funcionalidad en sí del proyecto pero que son indispensables para mantener una armonía con los demás componentes.

- **Descripción:** describe en que consiste el requisito.
- **Justificación:** indica el motivo por el cual se añade este requisito.
- **Criterio de satisfacción:** indica cómo se puede comprobar que el requisito se ha alcanzado con satisfacción.
- **Prioridad:** muestra el grado de prioridad del requisito valorado como Bajo, medio o Alto.
- **Satisfacción del cliente:** indica el grado de satisfacción que provoca sobre el cliente el cumplimiento del requisito, valorado con una escala ascendente entre 1 y 5.
- **Insatisfacción del cliente:** indica el grado de insatisfacción que provoca sobre el cliente el cumplimiento del requisito, valorado con una escala ascendente entre 1 y 5.

3.4 Requisito de estilo de botón exportación

Requisito #1	**Estilo del botón de exportación**				
Descripción	El botón será incorporado en una pestaña donde ya hay varios botones, por tanto para ser validado tiene que reunir unas cualidad que ya reúnen los otros botones, para que a nivel visual sean todos del mismo estilo.				
Justificación	El responsable deberá validar si el nuevo botón cumple con las especificaciones con las que ya contaban los demás botones de exportación.				
Prioridad	Media	**Satisfacción del cliente**	3	**Insatisfacción del cliente**	5

Tabla 3.5 Requisito de localización del botón de exportación

Requisito #2	Localización del botón de exportar				
Descripción	Este nuevo modo de exportación además de tener el mismo estilo que los anteriores, debe estar alineado con el resto para ofrecer visualmente una homogeneidad.				
Justificación	El responsable del proyecto valorará si se encuentra alineado y si cumple los requisitos visuales para no dañar la imagen que se ofrece.				
Prioridad	Media	**Satisfacción del cliente**	3	**Insatisfacción del cliente**	5

4. Especificación

A partir del análisis de los requisitos debe elaborarse la especificación; en este apartado, se describe el funcionamiento del sistema desde el punto de vista los actores que lo utilizaran.

4.1 Actores

Administrador

Este actor, tiene la capacidad de manejar todas las compañías, todos los usuarios e incluso manejar el balance de créditos de una empresa y de sus proyectos. Tiene la capacidad de realizar cambios siempre que estos sean necesarios.

Enterprise admin

Este usuario tiene la capacidad de realizar cambios en todo lo que tenga que ver con su empresa. Desde asignar proyectos a usuarios o crear nuevos usuarios hasta manejar el balance de créditos de cada uno de sus proyectos para reasignar o traspasar créditos entre sus proyectos disponibles. Es el máximo responsable de los proyectos de la empresa, por encima de los *project manager*.

Project Manager

Puede trabajar en los proyectos de su empresa y editar las preferencias de estos. Otras de las características de este usuario son:

- Ver los usuarios asignados a cada proyecto.
- Editar los proyectos según las necesidades.
- Borrar proyectos cuando ya no son funcionales.
- Borrar proyectos de su workspace.

Usuario

Es el encargado principal de utilizar *kCycle.* Estos actores tienen que realizar un seguido de validaciones para poder exportar los casos de prueba. En un principio deberán escribir HU o seleccionar algunas disponibles. Una vez realizada esta acción, deben dibujar ***Flows[11]*** o seleccionar alguno existente para realizar la creación de un flujo de pruebas **Eend nd to End**.

Una vez creados o seleccionados estos *flows* se procederá a continuar. Al cambiar de pantalla, este usuario va a tener acceso a una *preview* de los casos de prueba unitarios hasta que se realice un pago, que entonces tendrá acceso a los casos de prueba End to End. Después puede exportarse al formato que sea necesario.

4.2 Diagramas de casos de uso

A continuación, se muestran los diagramas de los casos de uso de cada tipo de usuario de kCycle. Pueden verse las funciones que puede desempeñar cada usuario en función de los permisos que tiene asociados a la cuenta.

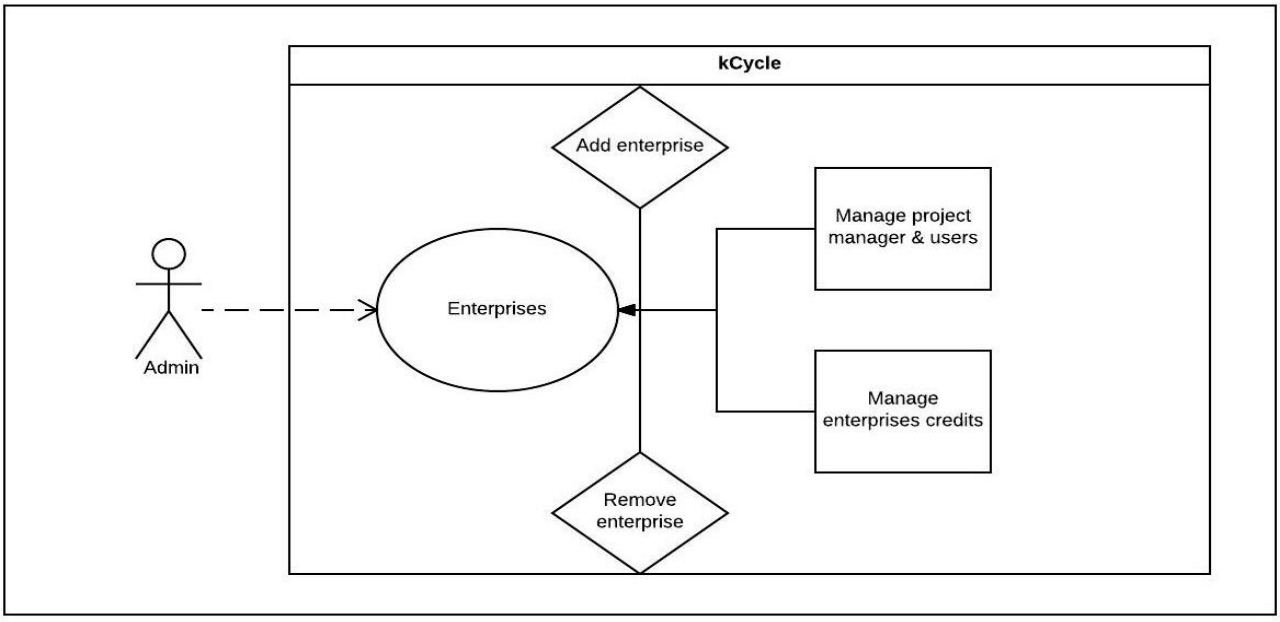

Figura 3 Casos de uso: Admin

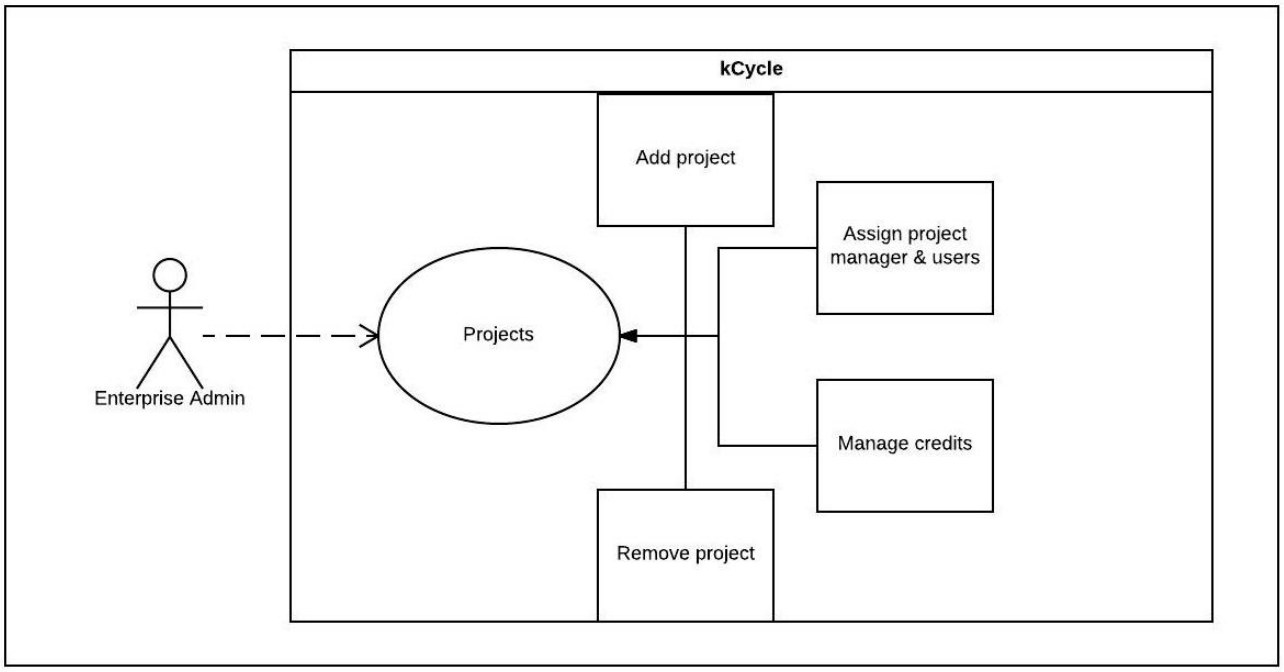

Figura 4 Casos de uso: Enterprise Admin

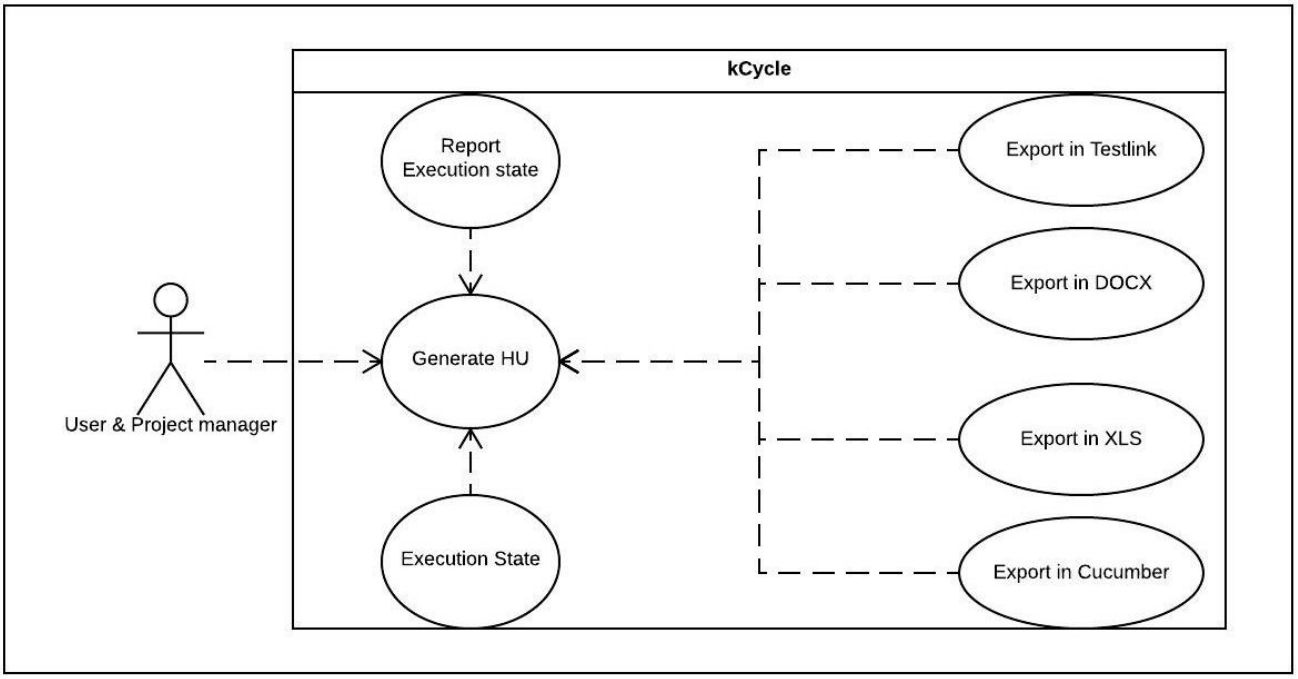

Figura 5 Casos de uso: Project Manager

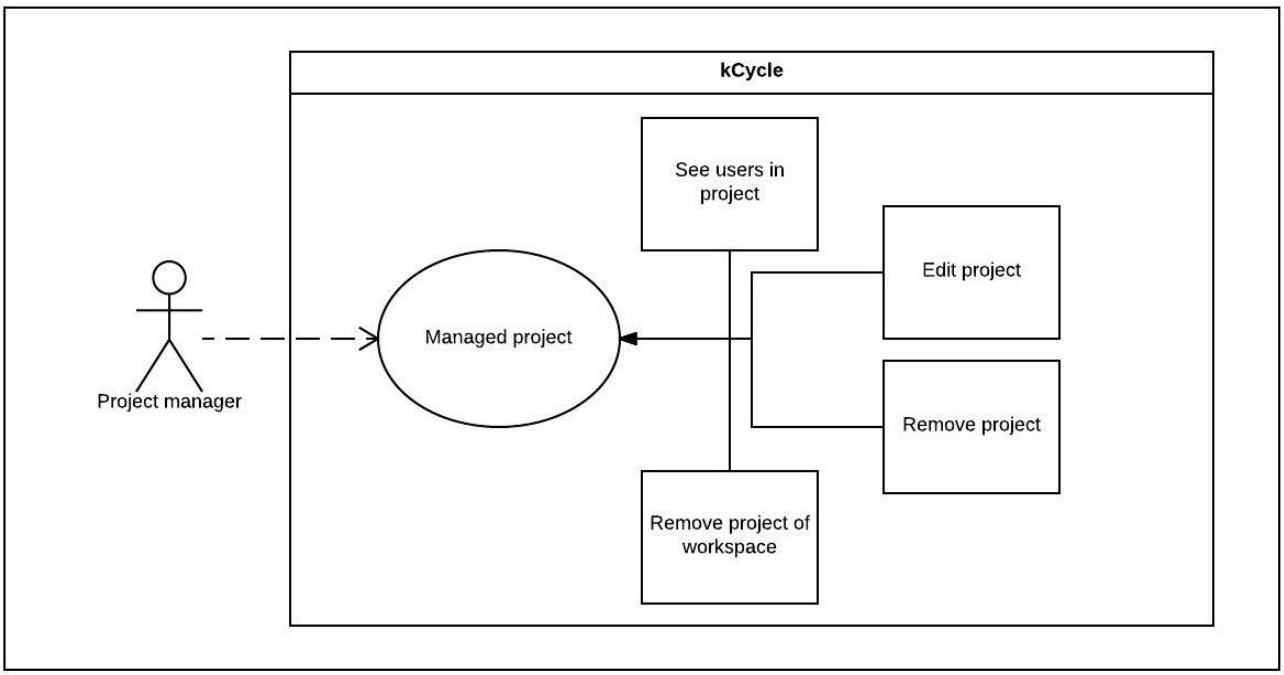

Figura 6 Casos de uso: User & Project Manager

5. Diseño e Implementación

En este apartado, va a tratarse el funcionamiento de *kCycle,* la arquitectura que se utiliza esta herramienta, el diseño de los nuevos componentes y la posterior implementación para cumplir con los requisitos especificados (ver sección 6).

5.1 Arquitectura lógica

Al hablar de arquitectura, hace referencia a los componentes del sistema y la manera que tienen de interactuar sobre la capa física para obtener el funcionamiento esperado de la herramienta.

Cómo el *framework* utilizado es Spring, la forma más útil de trabajar es con **Spring MVC**. Un proyecto que utiliza el modelo MVC con una estructura muy guiada.

El diseño del proyecto es separado en 3 capas, **Modelo**, **Vista** y **Controlador**. Estas capas van a ser descritas y ya que cada una tiene una finalidad distinta y posteriormente se definirá su función dentro del proyecto.

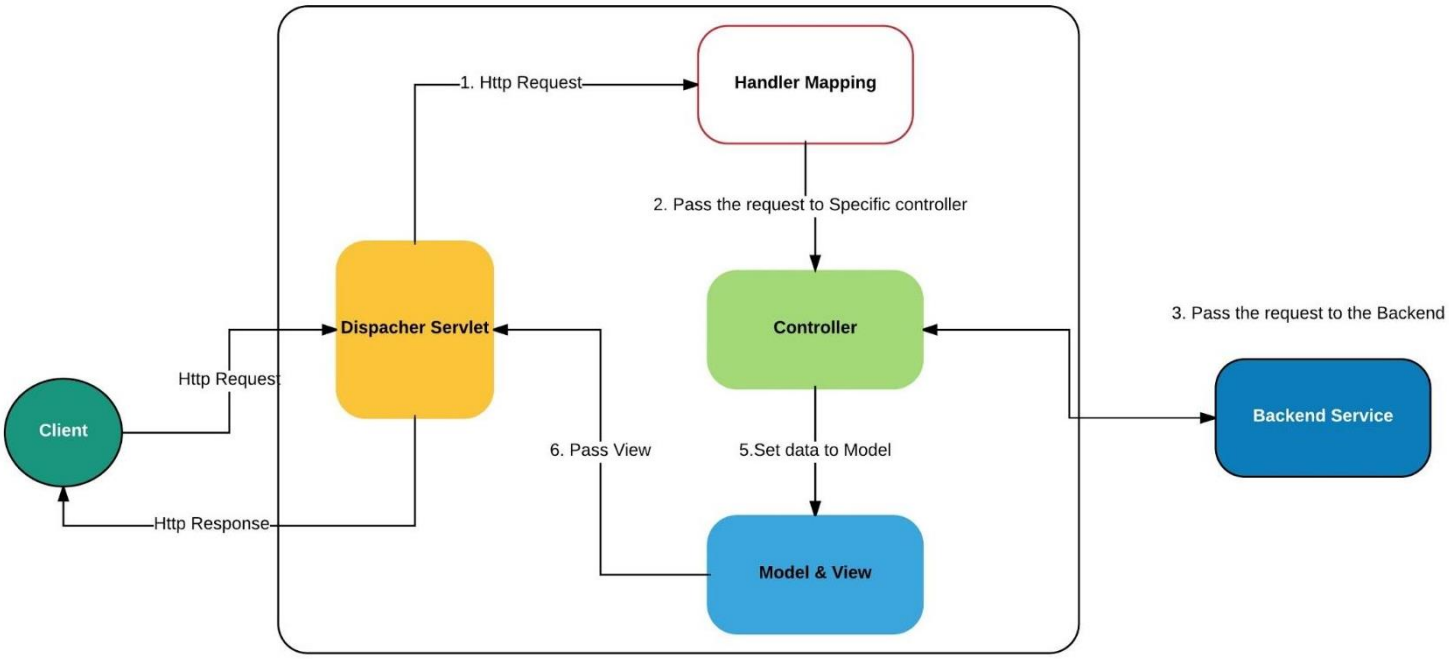

Figura 7 Arquitectura lógica de Spring

En la figura anterior podemos ver un esquema del funcionamiento interno de Spring en un **proyecto MVC**. A la izquierda del todo tenemos al cliente que después de realizar una petición al servidor, este realiza un ***mapping*** para saber a cuál de los controladores debe pasarle el control del ***Request*** realizado por el usuario.

Al encontrar el controlador indicado y pasarle la petición, este realiza en segundo plano acciones mientras que trabaja con el modelo y la vista indicados. Cuando ha obtenido el resultado del *Backend,* pasa los datos al modelo y la vista.

Cuando la vista y el modelo han realizado las peticiones necesarias, le devuelve la vista al ***Servlet*** y este responde al cliente con la vista.

La siguiente figura, aporta un conocimiento de la lógica del servidor, ya que no hay un servidor para cada aplicación, están en un solo servidor de manera física aunque de forma lógica parece que residan en diversos puntos. Gracias a la lógica creada, pueden coexistir sin colisiones ni errores ya que cada servicio se aloja en una red distinta.

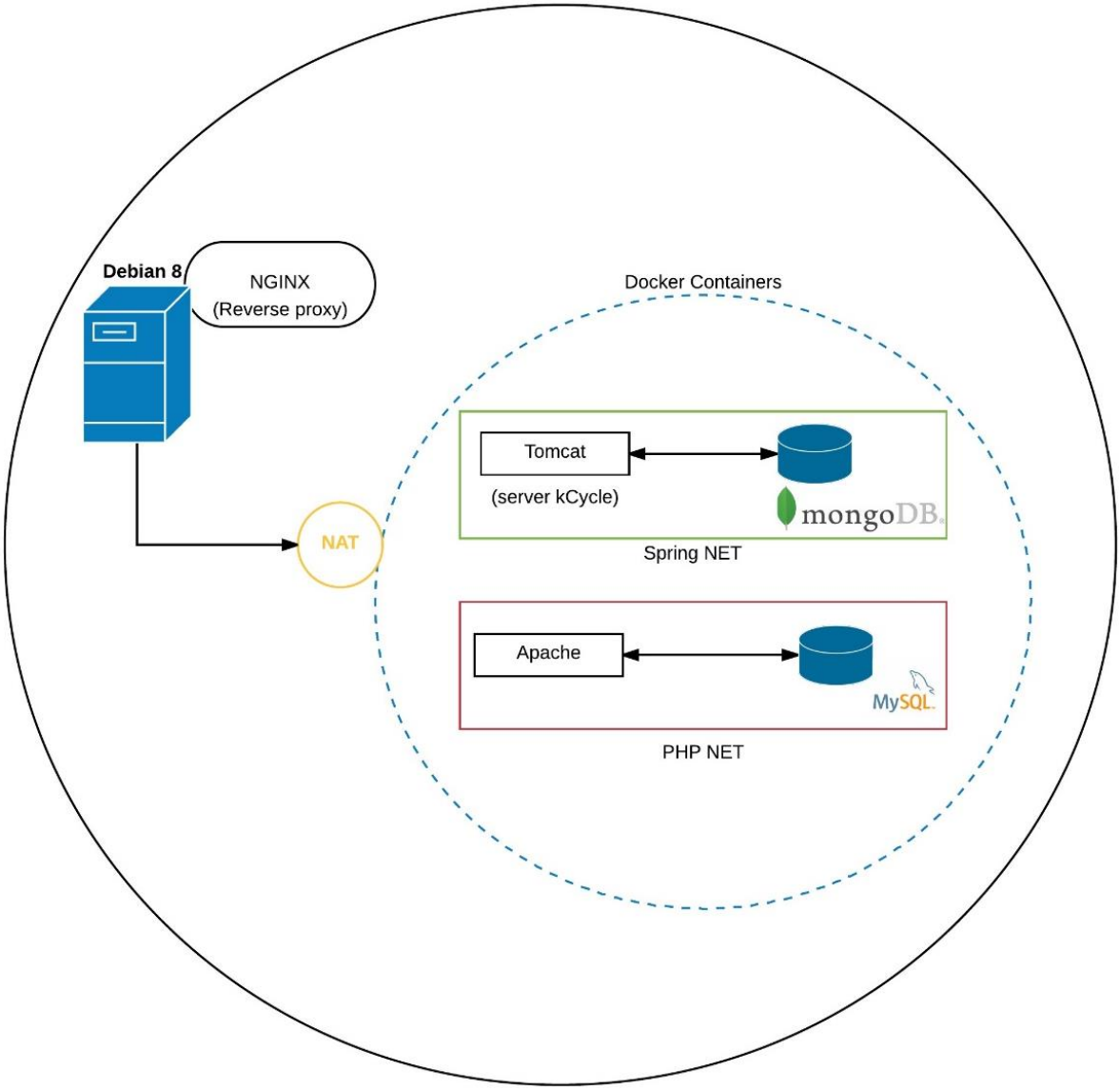

Figura 8 Arquitectura lógica del servidor de kCycle

Tabla 5.1 Ventajas e inconvenientes del uso de MVC

Ventajas
• Cada capa es completamente reemplazable sin que esto afecte al funcionamiento del resto de capas. • Los cambios en las capas sólo afectan a esa capa en concreto sin modificar las demás. • Sistema completamente escalable en caso de aumentar la complejidad.
Inconvenientes
• Menor eficiencia por la comunicación entre capas • Funcionamiento más complejo • Para probar el funcionamiento del sistema existe mayor dificultad, por la cantidad de elementos que intervienen

- **Modelo:** Contiene las funcionalidades relacionadas con el Modelo de datos. Se trata del acceso y manipulación de Bases de Datos y archivos referentes al proyecto.

- **Vista:** Se basa en el aspecto visual/gráfico, implica directamente el entorno gráfico que utilizará un usuario en la aplicación.

- **Controlador:** Capa intermedia entre el entorno gráfico (Vista) y el modelo ("Modelo"), coordina las peticiones del usuario para llevar a cabo las acciones deseadas.

5.2 Arquitectura física

La capa física de este proyecto se basa en una estructura cliente-servidor donde los usuarios utilizarán la red para conectarse con la herramienta alojada en un servidor.

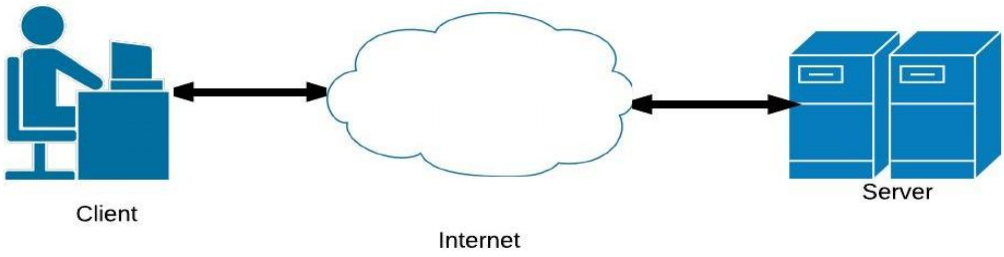

Figura 9 Arquitectura física de kCycle simplificada

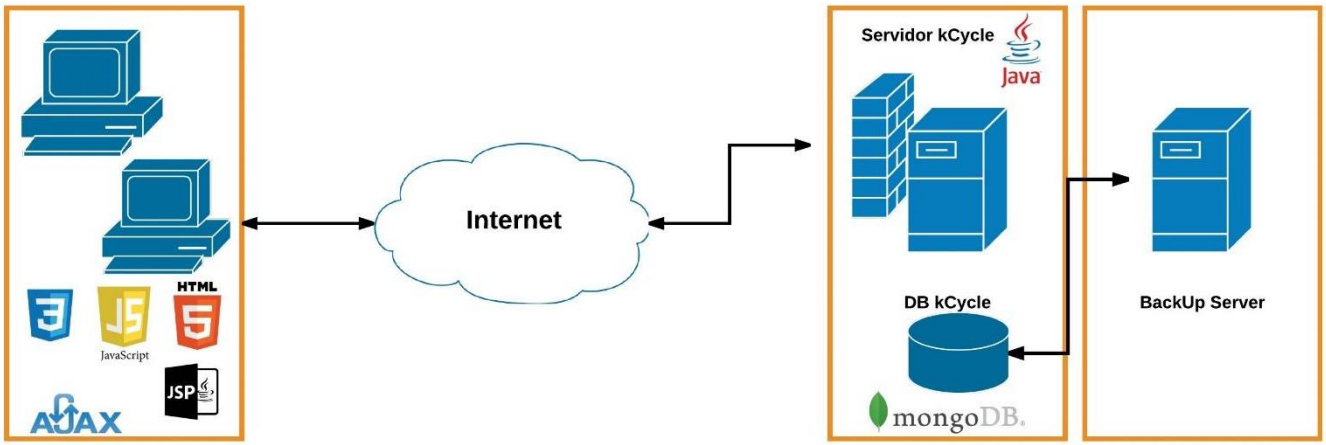

Figura 10 Arquitectura física de kCycle

5.3 Capa Modelo

En el modelo encontramos alojados los datos de kCycle. Aquí podemos encontrar las entradas de cada usuario desde los proyectos que tiene asociados a su cuenta hasta las ejecuciones y comentarios realizados.

Utilizando **Mongo DB** como central de datos, estos se encuentran encriptados para proteger la información de cada usuario y no poner en riesgo la confidencialidad los clientes de la herramienta.

5.3.1 *Gestión de datos para exportación a Cucumber*

Los datos para poder realizar la exportación a Cucumber se obtienen a partir de las selecciones del usuario. A partir de las HU que selecciona el usuario y de los *flows* que crea, se crean listas de los Test Cases que ha creado. A partir de estas listas, obtenemos las HU y los criterios de aceptación. Las listas son recorridas escribir en formato Gherkin los archivos FEATURE que crearan las clases JAVA posteriormente.

5.4 Capa Vistas

En *kCycle,* se utilizan muchas vistas para las diversas pantallas a las que puede acceder un usuario. Utilizando una estética representativa de SOGETI, para asegurar que los usuarios asimilen perciban los colores como un símbolo de calidad y asociarlo a la corporación.

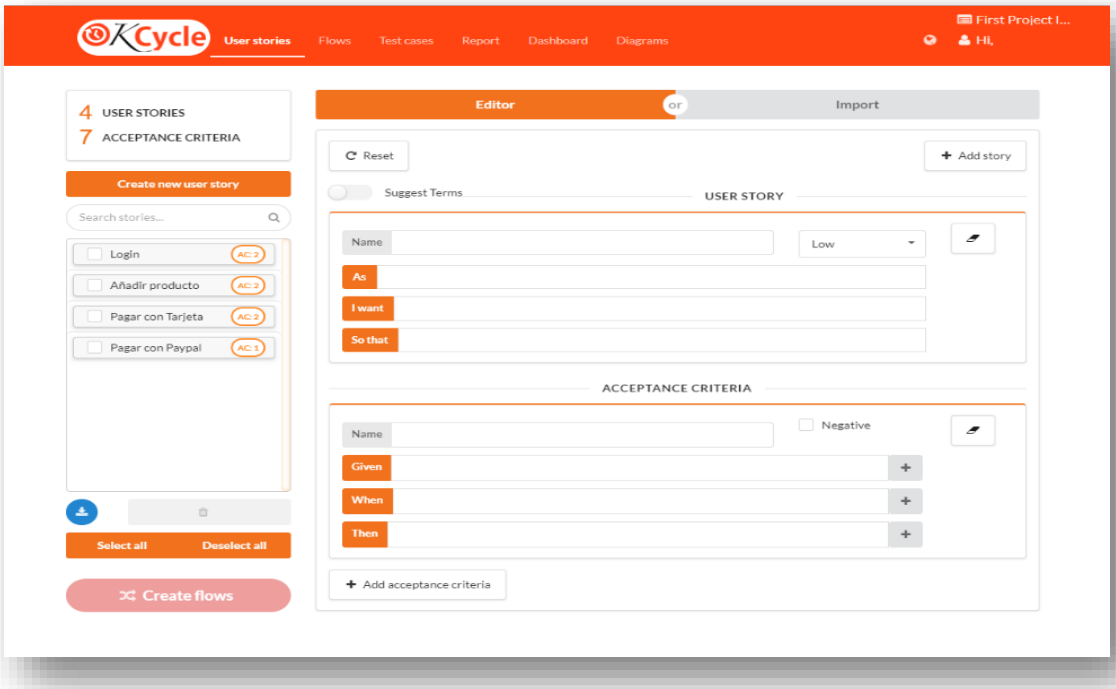

Figura 11 Vista principal de kCycle

La figura anterior, es una de las pantallas de la herramienta kCycle, con una estética clara que utiliza los colores corporativos. Una estética minimalista para no cargar la vista del usuario donde solo están presentes los elementos indispensables.

5.4.1 *Diseño botón de exportación a Cucumber*

Siguiendo el patrón que ya había visualmente, la creación del botón fue a través de Semantic-UI, un framework de diseño en el que puedes utilizar la solución más conveniente a tu sistema con un gran abanico de posibilidades. Los cambios que quieres introducir se hacen al introducir una clase.

Las formas y diseños de los elementos abarcan muchas formas y tipos. Aunque el diseño estaba preestablecido, se propusieron cambios para la estética, finalmente se decidió por dejar el diseño de los botones tal como estaban.

```
<div class="left_elements">
    <a id="export_features" class="ui fluid labeled blue icon button" download="TestCasesCucumber.zip" href="testCases/exportFeatures">
        <i class="download icon"></i>
        <fmt:message key="testcases_export_features" />
    </a>
</div>
```

Figura 12 Definición del botón de exportación en Cucumber

En la figura anterior puede verse la clase utilizada para la creación del botón. Utilizando una semántica clara, Spring entiende que se trata de un elemento predeterminado con una estética clara.

La figura también nos muestra que el proyecto va a descargarse en un archivo ZIP, que ayudará a comprimir el proyecto y descargarlo en un solo archivo. La descarga se realiza a través de la llamada al controlador correspondiente, que en este caso se trata de ‘exportFeatures’, alojado en la clase ‘testCases’.

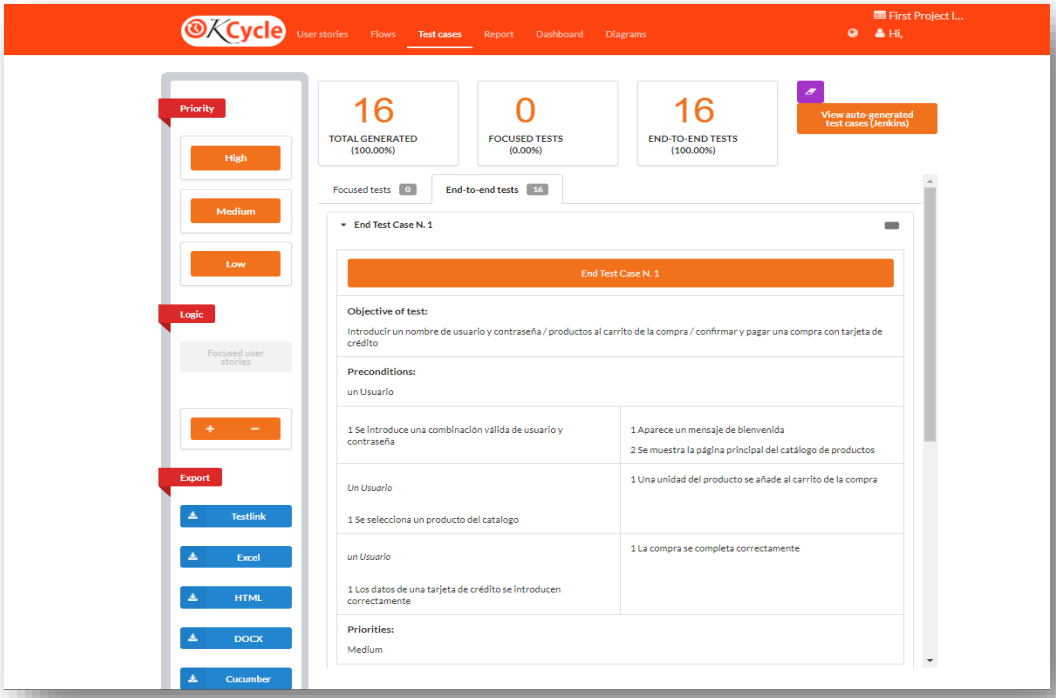

Figura 13 Botón de exportación de Cucumber implementado

Gracias a la interacción con la herramienta, el usuario realiza diferentes peticiones ya sean de datos o de acciones, para así comunicarse con el servidor y poder realizar las acciones que requiera.

5.5 Capa controlador

Esta Capa, realiza la **comunicación** entre las vistas y el modelo de datos. Cada vez que recibe una interacción del usuario (ya sea entrada de datos, o el pulsar un botón), el **controlador** entiende las peticiones del usuario.

Cuando ha recibido el *input*, el controlador realiza las acciones necesarias y recurre al **modelo** para recoger datos o incorporar de nuevos en la base de datos cuando sea preciso. Esta capa intermedia entre las vistas y los datos es **imprescindible** para el correcto funcionamiento de la herramienta ya que tiene que recoger correctamente los datos necesarios para realizar acciones con ellos. Una mala manipulación de los datos o una incorrecta llamada a un método tendrá como finalidad un mal funcionamiento de la herramienta.

5.5.1 *Diseño del controlador que genera los datos de exportación*

En este apartado, se trata el diseño del controlador que genera el proyecto. Son tratados los diferentes pasos que se han realizado, con los detalles necesarios para poder comprender como ha sido creado el controlador.

Al ser un controlador que va a responder con un archivo, hay que aplicar **Hooks[12]** propios de Spring. Los hooks utilizados son:

- **@ResponseBody:** Este hook nos sirve para decirle al controlador que vamos a devolver exactamente la respuesta en ese método, ya que si no buscaría una vista que no existe y el servidor devolvería un error 404

- **@GetMapping:** Nos sirve para devolver el elemento a descargar. El botón de exportación tiene que realizar una descarga y esta se realiza a través de una petición GET en el controlador. Normalmente se usa para devolver un JSON en formato String, aunque en este caso le damos una nueva utilidad.

Para poder acabar de asegurar que la respuesta será la correcta, el método creado es del tipo HTPP Entity. Este tipo de respuesta, devuelve una cabecera HTTP y el archivo ZIP con el proyecto de Cucumber. Para la creación del proyecto en sí, pasamos por diversos métodos que son llamados desde el controlador. Los métodos utilizados son:

- **Create_Dirs:** En este método hacemos la creación de los diferentes directorios del proyecto. Se empieza creando el directorio principal, después las diferentes carpetas que surgen a partir de la raíz.

- **createprojectFile:** De aquí surge la creación del archivo '.project'. Este es el archivo principal para que el IDE utilizado en el desarrollo de la automatización entienda que queremos importar un proyecto. En este archivo está la definición del nombre del proyecto, de los compiladores que van a utilizarse que en este caso son de MAVEN y de JAVA.

- **createFeatureFiles:** Cuando se realizada la llamada a este método, se crean los focused FEATURES files. Estos archivos son las HU que selecciona el usuario dentro del sistema de kCycle.

  La manera correcta para crearlos, es a través de los datos generados por kCycle a medida que el usuario va avanzando en el uso de la aplicación.

  Al tratarse de las HU unitarias, no habría ninguna necesidad de generar un hilo de ejecución ya que los datos ya han sido generados.

- **getEndtoEnd:** A diferencia que en el caso anterior, aquí es necesario crear un hilo de ejecución para generar los archivos End to End.

- **createPOM:** Este archivo se trata de un XML donde se reúnen las dependencias que tiene cada proyecto. Por tanto, se crea un archivo pom.xml en la raíz del proyecto.

  Una vez se ha creado el archivo, hay que seguir una estructura concreta para la correcta creación del archivo. Esto es justo lo que realiza este método, escribe las dependencias indispensables para trabajar en proyectos de *Cucumber* correctamente.

- **createClasspath:** Este método nos sirve para poder crear un archivo *'.classpath'*. Este archivo avisa al sistema que librerías de JAVA va a utilizar el proyecto. La forma de crear este tipo de archivos es similar a la del pom con la diferencia de que este no es un archivo XML pero eso no tiene dificultad ya que tan solo hay que especificarlo al inicio, cuando se crea el proyecto. Este tipo de archivos es bastante similar en cualquier proyecto así que para su correcta creación es necesario observar en cualquier otro proyecto la estructura de la este archivo e intentar copiarla.

- **readMe:** La creación de este archivo es necesaria para poder guiar al usuario en caso de no saber correctamente cómo funciona el proyecto. Es una pequeña guía para iniciar a quien no tenga conocimiento del tema.

- **createSteps:** método diseñado para crear las clases JAVA. A partir de la creación de los archivos FEATURE este método realiza un *parsing* para leer los criterios de calidad del las HU. Cuando empiezan los criterios de calidad de las HU utilizando las Keywords de Gherkin, se guardan en un buffer para después escribirlo en un archivo *'.java'*.

5.6 Implementación

La etapa de implementación, es en la que se aplican los requisitos definidos en la etapa de especificación y la de diseño para la obtención de las funcionalidades definidas como objetivo. En este apartado se detalla el resultado y el proceso de implementación de las funcionalidades, así como los detalles técnicos de las mismas.

5.6.1 *Proyecto de Cucumber autogenerado*

Cuando se han creado todos los archivos necesarios, estos son guardados en sus respectivas carpetas que acaban siendo alojadas en la raíz del proyecto. A partir de ahí empieza el proceso de descarga. El proyecto es comprimido con todos sus archivos en un ZIP que se aloja en una carpeta temporal del servidor de Tomcat.

El proceso de descarga finaliza cuando el controlador devuelve una cabecera HTTP y el proyecto comprimido. Cuando se ha descargado, el controlador comprueba si existe una carpeta con el nombre del proyecto y posteriormente un archivo ZIP. Si esto es así, borra ambos archivos para no malgastar espacio en la carpeta del servidor.

5.6.2 *Utilización de datos para creación de archivos FEATURE*

Cuando se ha definido como va a crearse el método que creará los archivos FEATURE se procede a buscar la manera de conseguir esos datos para realizar la implementación de la funcionalidad diseñada. Observando el funcionamiento de kCycle y el flujo que debe seguir el usuario, es posible obtener los datos de las historias de usuario seleccionadas por el usuario y de los *flows* generados.

Con esos datos conseguidos, recorremos la lista de HU que se han generado a partir de seguir el flujo de desarrollo. Escribimos en formato Gherkin los datos para crear los archivos FEATURE que posteriormente ayudarán a su vez a escribir las clases JAVA necesarias.

5.6.3 *Clases JAVA autogeneradas a partir de archivos FEATURE*

Una vez diseñado el método, la implementación total viene seguida de conseguir imitar el estilo de un método del tipo **void**.

Además, se intenta emular una clase JAVA creada en cualquier proyecto, hay que escribirle el *package* al que está asociado esta clase, definido en la reunión estructura de carpetas realizada anteriormente.

Las *keywords* escritas en los archivos FEATURE, en la clase JAVA nos sirven para asociar los criterios de aceptación (CA) descritos en Gherkin a los métodos escritos en la clase JAVA. Gracias a esto, el *runner* sabe leer los criterios de aceptación que hay que testear y asociarlos a los métodos creados.

Estos métodos posteriormente deben utilizarse para escribir en Selenium los pasos que va a seguir el driver creado.

6. Plan de pruebas

El *testing* de este proyecto, se ha hecho a nivel de funcionalidad. La intención de estas pruebas es validar que el sistema cumple con los requisitos descritos en la fase de especificación.

6.1 Pruebas Funcionales

A continuación definimos el plan de pruebas que establece si los requisitos funcionales han sido alcanzados; aunque las pruebas muestran escenarios positivos, también se han hecho validaciones realizando las mismas acciones para los casos negativos, en los que los criterios de aceptación no son cumplidos.

Este plan de pruebas, se ha ejecutado en cada cambio introducido una vez finalizada la segunda iteración.

Tabla 6.1 Caso de prueba de generación y descarga del proyecto

Caso de Prueba #1	**Generar y descargar proyecto**
Acciones	1. El usuario crea historias de usuario y las selecciona. 2. El usuario genera un flow, lo guarda, lo selecciona y continúa. 3. El usuario entra en la pestaña Test Cases y le da al botón 'Exportar en Cucumber.
Criterios de aceptación	o El sistema, nos realiza la descarga del proyecto en un archivo ZIP, sin ninguna indicación de que este está dañado o de que no es posible abrirlo

Tabla 6.2 Caso de prueba de visualización de archivos

Caso de Prueba #2	Visualizar archivos del proyecto
Acciones	1. El usuario abre Eclipse e importa el proyecto después de haberlo descomprimido. 2. El usuario puede ver la estructura de carpetas una vez realizada la importación. 3. El usuario puede seleccionar un archivo y editarlo, siendo estos cambios reconocidos por el editor.
Criterios de aceptación	o Eclipse, nos da la posibilidad de guardar los cambios editados y reconoce los objetos creados, haciendo sugerencias durante la edición.

Tabla 6.3 Caso de prueba de comprobación de concordancia entre HU y métodos

Caso de Prueba #3	Comprobar concordancia entre HU y métodos JAVA
Acciones	1. El usuario abre Eclipse e importa el proyecto después de haberlo descomprimido. 2. El usuario puede ver los archivos tipo Gherkin y las clases JAVA. 3. El usuario puede comprobar que los métodos tienen el mismo nombre que los criterios de aceptación de las historias de usuario.
Criterios de aceptación	o Los métodos son creados a partir de las HU ya que podemos ver que apuntan directamente a la definición del criterio de aceptación de la historia de usuario a la que pertenece.

Tabla 6.4 Caso de prueba de comprobación de la correcta creación del POM

Caso de Prueba #4	Comprobar creación de POM
Acciones	1. El usuario abre Eclipse e importa el proyecto después de haberlo descomprimido. 2. El usuario abre el archivo POM. 3. El usuario puede ver que las dependencias del archivo han sido creadas correctamente.
Criterios de aceptación	o Al abrir el archivo POM, las dependencias definidas son las esperadas cuando se ha definido el método que crea el archivo en cuestión.

Tabla 6.5 Caso de prueba de comprobación de correcta creación del runner

Caso de Prueba #2	Comprobar correcta creación del runner
Acciones	1. El usuario abre Eclipse e importa el proyecto después de haberlo descomprimido. 2. El usuario abre el *runner* con la intención de visualizarlo. 3. El usuario comprueba que el resultado es el esperado, ya que en el *runner* está definido el alojamiento de las *Step definitions* y las HU.
Criterios de aceptación	o Al abrir el *runner* puede observarse que nos indica donde pueden hallarse los criterios de aceptación.

Adicionalmente a estas pruebas, una vez el funcionamiento es correcto según los criterios de aceptación definidos, esta nueva funcionalidad ha sido probada con diferentes proyectos en kCycle, para comprobar que las HU eran correctamente extraídas tanto para los **Casos de prueba Unitarios**, como para los casos de tipo **End To End**.

6.2 Gestión de defectos e informes

En un contexto de desarrollo colaborativo, es imperativo mantener un control preciso de la situación, de los errores que puedan ir apareciendo a medida que añadimos nuevas funcionalidades o al arreglar las existentes.

La herramienta utilizada para ello es **GitHub**, en el repositorio se añade una nueva entrada en la pestaña Issue. El *tester* encargado de probar la nueva funcionalidad, expresará con imágenes, videos o la forma que considere oportuna para mostrar al desarrollador donde se encuentra el fallo y los pasos realizados para llegar a obtener ese error.

Por lo general, se optará a crear una nueva rama para paliar esos errores reportados. Esta nueva rama nace a partir de la rama principal de desarrollo y por tanto el desarrollador seguirá manteniendo la última versión del proyecto.

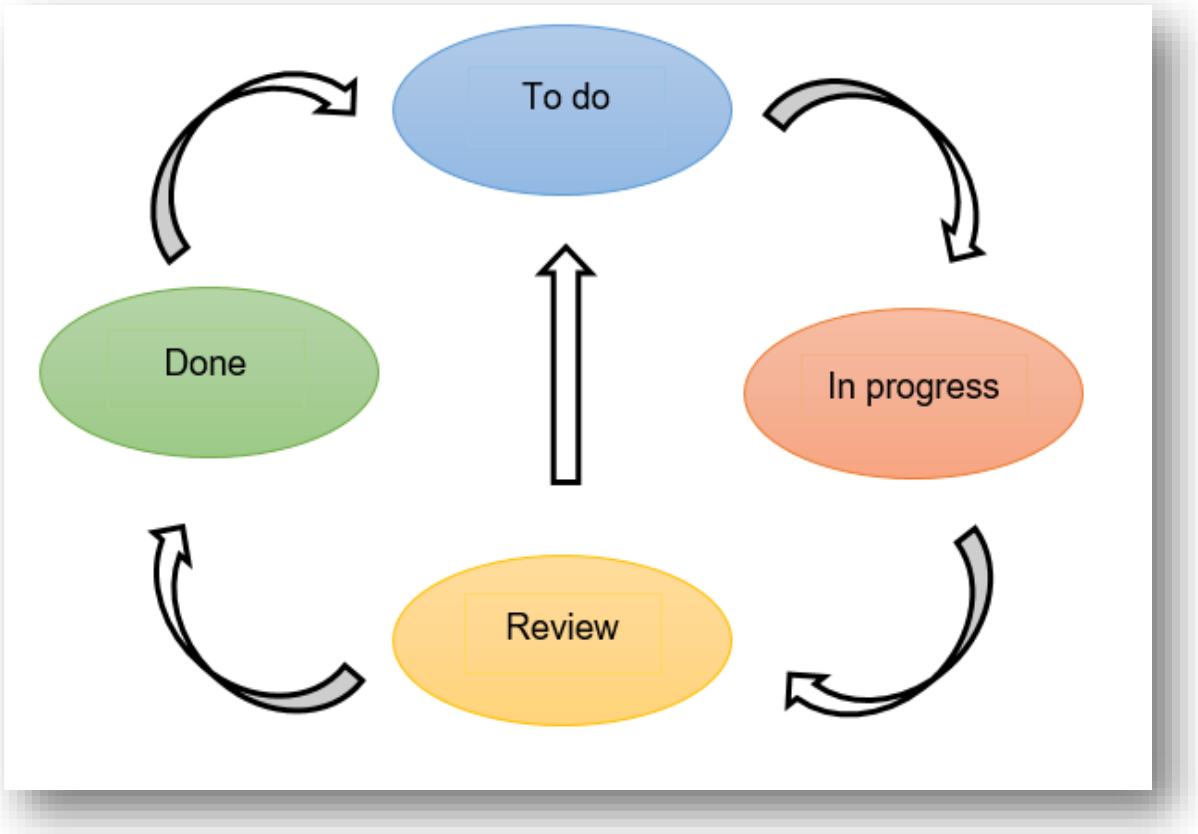

Figura 14 Definición del proceso de defectos

En este proyecto, cada defecto abierto tiene un ciclo de vida en el cual el estado en el que se encuentra va cambiando a medida que se trabaja sobre ellos. Los estados posibles que puede adoptar son los siguientes:

- **To do:** El defecto aún está pendiente de solucionar.
- **In progress:** La solución del defecto está en desarrollo.
- **Review:** Revisión de la solución del defecto.
- **Done:** Defecto solucionado correctamente.

Cabe resaltar, que los defectos pueden volver a surgir aunque se haya dado una solución correcta o que esta, esté en proceso de desarrollo, en cuyo caso, el defecto volvería al estado '**To do'**.

7. Gestión del proyecto

7.1 Planificación temporal

El proyecto, tiene una duración de cuatro meses y medio teniendo en cuenta que el inicio fue la gestión del proyecto y el final ha sido la redacción de esta memoria.

Esta planificación ha sido estudiada de manera que puedan cumplirse los plazos para el correcto desarrollo e integración de la nueva funcionalidad con la versión Beta de la aplicación, que debe ser utilizada por usuarios del sector para testear de manera externa el correcto funcionamiento de la herramienta,

7.2 Descripción de tareas

En este apartado se describe cada etapa realizada a durante la duración del proyecto.

Gestión del proyecto

En esta fase, se decide cómo va a realizarse el proyecto, el problema planteado debe ser analizado y especificar cuál va a ser el camino a seguir para llegar a una de las posibles soluciones.

Primera iteración: Mecanismo para la obtención de las HU y los requisitos

En esta primera iteración, se realiza un estudio previo del funcionamiento de kCycle. La finalidad de este software y el camino crítico que realiza para funcionar correctamente son los primeros temas a abordar.

Una vez resueltas estas dudas, se realiza un estudio para poder definir el mecanismo para la obtención de las Historias de Usuario y de los criterios de aceptación definidos por el usuario. **Sigue!!**

Segunda iteración: Implementación de la lógica en un proyecto de Cucumber

En esta segunda iteración, se recogen ideas de diversos desarrolladores sobre la planificación del proyecto de *Cucumber.* La distribución de las diferentes carpetas que intervienen en un proyecto juegan un papel importante, ya que una distribución óptima del contenido facilita la vida al usuario final de este proyecto, para hacerlo más intuitivo y asequible.

Una vez decidido, el siguiente paso consiste en generar toda la documentación necesaria para que sea entendido como un proyecto de Maven. Tres archivos son elementales para el correcto funcionamiento de este proyecto. El primer archivo necesario es el archivo que define el proyecto, en el que se define el nombre y otras propiedades.

Después iría el archivo Classpath, en este archivo son definidas las librerías de **JAVA** que van a utilizarse y que versión. Este archivo es necesario para poder utilizar correctamente las librerías sin que se hayan quedado obsoletas.

Para finalizar, el último archivo necesario sería el **POM** (Project Object Model). Este archivo contiene las dependencias a librerías externas a java (entre otras funcionalidades), como serían las librerías de **Selenium** o **Gherkin** por ejemplo. Teniendo estas dependencias nos aseguramos que el usuario de este proyecto no tenga que tener estas librerías ya que **Maven** se encarga de encontrar las dependencias y descargar las librerías que sean necesarias para el desarrollo del proyecto.

Tercera iteración: Implementación de la funcionalidad en entorno de producción

En esta tercera iteración, se realiza la implementación a un entorno de producción. Durante esta implementación, será necesario actualizar la versión del proyecto ya que el resto del equipo ha ido avanzando y corrigiendo errores.

Esta parte conlleva la corrección de la nueva funcionalidad ya que debe funcionar sin errores y sin ‘romper’ ninguna otra funcionalidad. Este paso se realiza con las **pruebas de funcionalidad**, vigilando los posibles errores y corrigiendo las partes que no cumplen con los requisitos.

Etapa final

Durante esta última etapa, se acabó de preparar tanto la memoria del proyecto como la presentación.

La memoria ha sido elaborada a medida que iba finalizando el proyecto. Durante la parte de desarrollo se han ido tomando notas y apuntes, como por ejemplo el análisis de los requisitos y diseño de cada una de las iteraciones. Una vez finalizado el desarrollo, se han juntado las notas y ha procedido a finalizar la memoria.

7.3 Recursos

Los recursos que se prevén en el desarrollo de este proyecto son:

Recursos humanos: Una sola persona con una dedicación en el proyecto de 20 horas semanales, asumiendo los siguientes roles:

- Analista
- Programador
- Tester

Recursos materiales: Un ordenador portátil donde se desarrolla en local, además de ser utilizado como servidor local para hacer pruebas. A medida de que el proyecto avanza, un servidor donde se aloja la versión de producción de **kCycle.**

Recursos *software*:

- Entorno de desarrollo para tecnología Spring: Eclipse IDE.
- Sistema de control de versiones: GitHub y Git para su gestión.
- Herramientas de documentación: Microsoft Word 2013, PowerPoint 2013.
- Gestor de versiones de documentos: Google Drive y One Drive.

7.4 Estimación del tiempo

Tabla 7.1 Estimación del tiempo de proyecto

Tarea	Dedicación (Horas)
Gestión de Proyectos (GEP)	**179**
Contextualización y alcance	35,5
Planificación temporal	12
Documento final	11
Presentación oral	10,5
Estudio previo sobre Cucumber	55
Estudio previo sobre Spring	55
Primera iteración: Mecanismo para la obtención de las HU y los requisitos	**110**
Especificación y diseño	25
Implementación e integración	68
Testing	17
Segunda iteración: Implementación de la lógica en un proyecto de Cucumber	**122**
Especificación y diseño	30
Implementación e integración	75
Testing	17
Tercera iteración: Implementación de la funcionalidad en entorno de producción	**112**
Especificación y diseño	30
Implementación e integración	65
Testing	17
Etapa final	**62**
Documentar y finalizar memoria	42

Preparación de la exposición oral	25
Total	**528**

8. Conclusiones

8.1 Consecución de los objetivos

Una vez finalizado el proyecto puede concluirse que se han conseguido los objetivos establecidos. Ahora, puede crearse por parte de los usuarios un proyecto de Maven utilizando el framework de *Cucumber*. Estos objetivos alcanzados ayudarán al usuario a poder automatizar sus sistemas, con todos lo archivos necesarios, desde las HU hasta las clases JAVA que dependen de estas. No obstante, no ha sido sencillo llegar a cumplir con los objetivos por los imprevistos que han surgido a medida que avanzaba el proyecto.

Cabe destacar, que aunque haya habido algún desviamiento temporal, esto no ha impedido finalizar el proyecto en la fecha estimada. El haber tenido presente en la planificación que podrían surgir problemas, ha permitido solventar los problemas que han ido surgiendo.

Las soluciones encontradas, aunque posiblemente sean mejorables, han sido estudiadas minuciosamente para poder cumplir con los requisitos del sistema implantados al inicio del proyecto. Para poder ser implantadas estas nuevas funcionalidades en kCycle siempre ha sido con el visto bueno del jefe de proyecto, quien se aseguraba que cumplian los requisitos.

8.2 Mejoras futuras

Aunque se han cumplido los objetivos establecidos, las funciones implementadas aceptan modificaciones o cambios que ayuden a mejorar la herramienta kCycle.

Ampliaciones:

- **Ampliación de dependencias POM:** Podría ayudar la ampliación de estas definiciones para dar al usuario más opciones por defecto.

- **Integración de plugins:** Desde el archivo POM podrían ponerse las dependencias por defecto de los plugin para la generación automática en HTML de las ejecuciones realizadas. Estas dependencias sólo serían efectivas si en el *Runner* hacemos las definiciones, si no tan solo tendríamos lo que nos ofrece **JUnit** por defecto.

Mejoras:

- **Tagging de los criterios de aceptación:** Actualmente, el tagging que utilizamos para diferenciar en el *Runner* los test que quieren lanzarse en una ejecución va numerado. Esta numeración va añadiendo un número a cada **CA** que va creando el algoritmo.

- **Nombre Archivos FEATURE:** Los archivos End To End van numerados desde el cero (0) hasta el número de test End To End que hayan. Podría encontrarse la manera de encontrar un nombre identificativo para cada prueba End to End.

- **Nombre Step Definition:** Los archivos JAVA que contienen las *Step Definitions* tienen un nombre genérico dependiendo si son *Focused Steps* o *End to End Steps,* diferenciadas por subíndices numéricos. Podría buscarse la manera de escoger un nombre identificativo que esté enlazado a las HU y que el usuario sepa identificarlo.

8.3 Valoración personal

Poder realizar este proyecto, me ha ayudado a poder crecer tanto a nivel personal como profesional. Por lo que refiere al nivel personal, es una referencia a saber establecer requisitos en los proyectos, a tener una visión crítica del trabajo y sobre todo a comprender mis errores e intentar buscar soluciones a estos por mi propia cuenta. A nivel profesional he podido crecer ya que he aprendido a trabajar con requisitos, a tener fechas límites y trabajar con presión en un entorno laboral. Además, a nivel técnico he aprendido y he obtenido un perfil más técnico del que tenía inicialmente.

No ha sido sencillo poder alcanzar los objetivos ya que he tenido que leer mucho sobre desarrollo y sobre el funcionamiento de Cucumber. Al inicio del proyecto, la idea me parecía muy buena pero era difícil de creer que pudiese hacer yo solo la implementación y ejecución de una funcionalidad tan complicada en un proyecto tan importante.

En definitiva, estoy satisfecho con el trabajo realizado, con los resultados obtenidos y sobre todo con el aprendizaje obtenido al final de la realización de un proyecto tan ambicioso.

9. Glosario

[1] Sogeti: Empresa con expertos en Gestión del Cambio y con una amplia experiencia en Digital Assurance & Testing y en el Desarrollo de Soluciones bajo tecnología Microsoft.

[2] Pruebas manuales: Pruebas que realiza un tester simulando que es un usuario pero con la diferencia de que este sigue unos pasos definidos por un Test case.

[3] Keywords: Sintaxis de Gherkin para Cucumber y Selenium. Se utiliza para definir los pasos que van a seguirse en un criterio de aceptación. Las palabras clave son entre otras: Given, When, Then, And, But y Background.

[4] Criterios de aceptación: Condiciones que un producto de software debe satisfacer para ser aceptado por un usuario, cliente o stakeholder.

[5] Casos de prueba end-to-end: Casos de prueba que siguen un flujo lógico. Surgen a partir de casos de prueba unitarios donde solamente se prueba la funcionalidad testeada. Los casos end-to-end siguen un flujo lógico de una aplicación.

[6] Behaviour-Driven Development: Es un proceso de desarrollo software principalmente creado para testing. **BDD** busca un lenguaje común para unir la parte técnica y la de negocio, y que sea desde ese lenguaje común desde donde arranque el Testing y, desde ahí, el desarrollo.

[7] Historias de usuario: Representación de un requisito escrito en una o dos frases utilizando un lenguaje común. Se utilizan comúnmente en las metodologías ágiles. Estas HU deben ser limitadas y escritas en pequeñas notas.

[8] Selenium: Conjunto de librerías que se utilizan para adquirir el control de distintos drivers. Al obtener el control, podemos realizar acciones de forma automática simulando que es un usuario.

[9] Framework: Entorno de trabajo que representa una arquitectura de software que modela las relaciones generales de las entidades del dominio.

[10] Maven: Es una herramienta de comprensión y manejo de software basado en el concepto de POM. Pueden manejar builds, reportar y crear documentación.

[11] Scrum: Nombre con el que se denomina un proceso de creación ágil. Adopta una estrategia de desarrollo progresivo e incremental en vez de una estructura marcada y una ejecución completa.

10. Referencias

[1] A. Tort, «Testing and Test-Driven Development of Conceptual Schemas,» 2012.

[2] S. Meyers, «Effective JavaScript 68 Specific ways to harness the Power of JavaScript,» 2013.

[3] J. Bloch, «Effective Java».

[4] R. RV, Spring Microservices, 2016.

[5] «GitHub,» [En línea]. Available: https://github.com/SeleniumHQ/selenium.

[6] «ToolsQA,» [En línea]. Available: http://toolsqa.com/cucumber/cucumber-tutorial/.

[7] «SeleniumHQ,» [En línea]. Available: http://www.seleniumhq.org/.

[8] «GitHub,» [En línea]. Available: https://github.com/cucumber/cucumber/wiki/Gherkin.

[9] «Software Testing Help,» [En línea]. Available: http://www.softwaretestinghelp.com/cucumber-bdd-tool-selenium-tutorial-30/.

[10] «Cucumber,» [En línea]. Available: https://cucumber.io/docs.

[11] «GenBETA,» [En línea]. Available: https://www.genbetadev.com/metodologias-de-programacion/bdd-cucumber-y-gherkin-desarrollo-dirigido-por-comportamiento.